Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

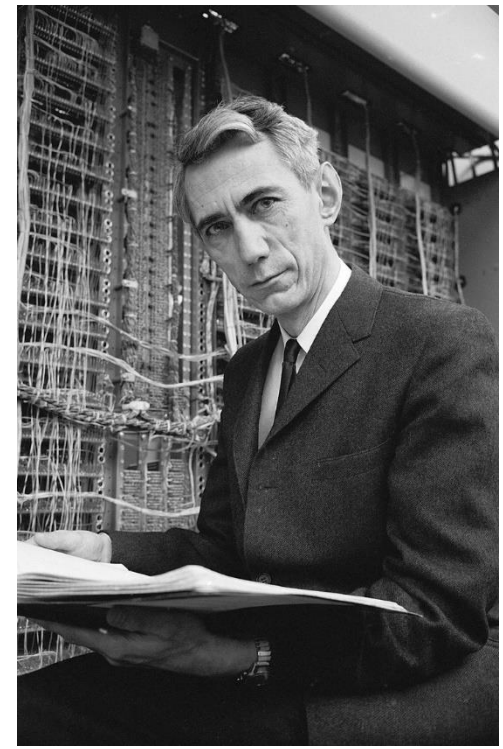
IIC2343 – Arquitectura de Computadores

Operaciones Aritméticas y Lógicas

Profesor: Hans Löbel

A fines de los 30s, las máquinas analógicas dominaban la computación

- Claude Shannon (padre de la teoría de la información) estaba convencido que esta no era la mejor solución.
- Planteo una estrategia de tres pasos para diseñar un computador más eficiente, que finalmente resultaría siendo el computador digital.
- Desarrolló estas ideas en su tesis de magister, a los 19 años.



Como cambiar de analógico a digital

- Un sistema numérico eficiente.
- Mecanismo para operar eficientemente sobre este sistema numérico.
- Procedimiento para transformar lo anterior en elementos físicos.

Como cambiar de analógico a digital

- Un sistema numérico eficiente.
- Mecanismo para operar eficientemente sobre este sistema numérico.
- Procedimiento para transformar lo anterior en elementos físicos.

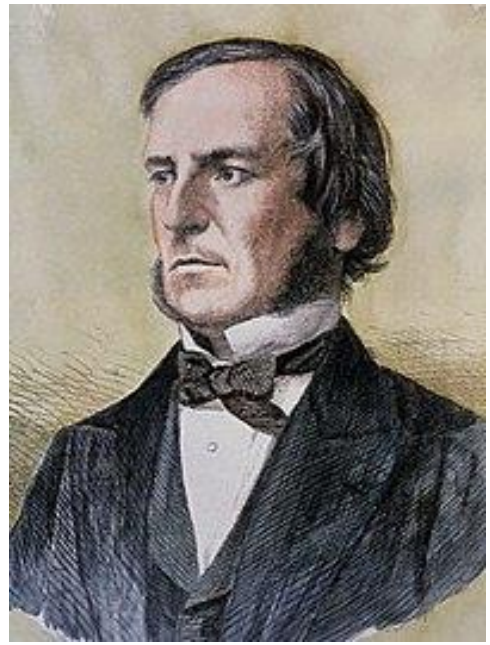
Como cambiar de analógico a digital

- Un sistema numérico eficiente.
- Mecanismo para operar eficientemente sobre este sistema numérico.
- Procedimiento para transformar lo anterior en elementos físicos.

Lógica Booleana es la solución

- En 1937, Shannon notó la relación entre la lógica booleana y la operación de números binarios.

- La lógica booleana permite analizar y operar proposiciones que pueden tomar sólo 2 valores: verdadero (1) o falso (0).



Una fórmula de lógica booleana tiene 2 componentes principales

- Proposiciones lógicas o variables: verdadero o falso

  A = la luz está prendida

  B = está nublado

- Conectivos lógicos

  Permiten unir las variables, análogo a operadores (+,-,…).

  Se definen usando tablas de verdad.

A	B	A and B
F	F	F
F	V	F
V	F	F
V	V	V

A	B	A or B
F	F	F
F	V	V
V	F	V
V	V	V

A	not(A)
F	V
V	F

Álgebra booleana permite definir sentencia lógicas complejas

- Bastan AND, OR y NOT para representar cualquier tabla de verdad de conectivos binarios.
- Para definir sentencias complejas, basta con conectar múltiples variables y operadores.
- Por ejemplo:

$$A \oplus B = (\neg A \wedge B) \vee (A \wedge \neg B)$$

Álgebra booleana permite definir sentencias lógicas complejas

- Lo mejor es que podemos representar operaciones aritméticas fácilmente

A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = A \oplus B$

$C = A \wedge B$

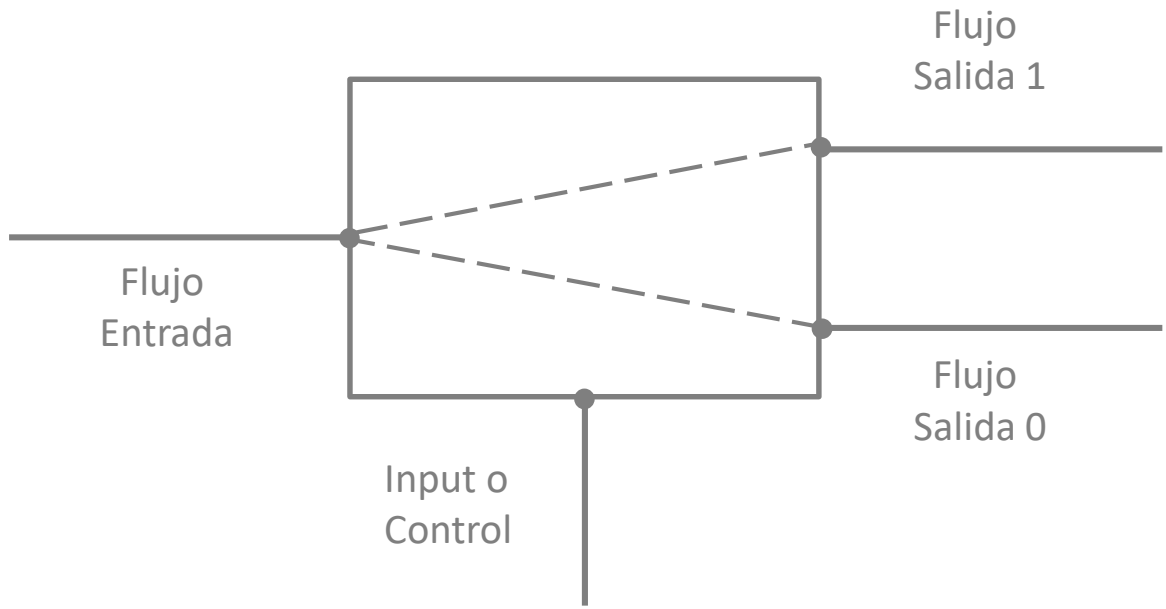
Como cambiar de analógico a digital

- Un sistema numérico eficiente.
- Mecanismo para operar eficientemente sobre este sistema numérico.
- Procedimiento para transformar lo anterior en elementos físicos.

Podemos implementar físicamente los componentes de la lógica booleana

- El gran aporte de Shannon, fue describir implementaciones físicas de AND, OR y NOT.
- Por cada uno de estos conectivos definió una compuerta lógica.
- Luego, implementó cada compuerta mediante relés.

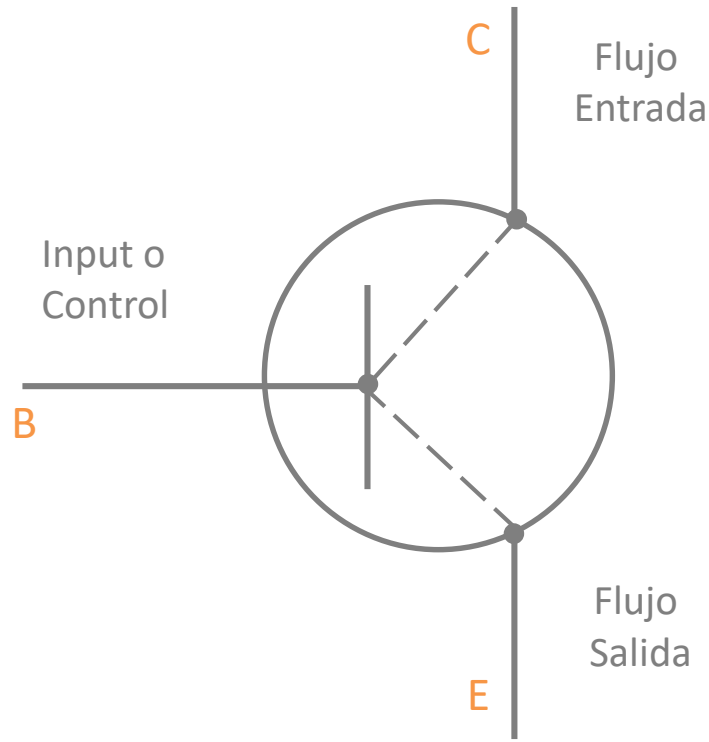
Relé



Sin flujo de control => Input == 0 → Flujo por salida 0 => Output 0

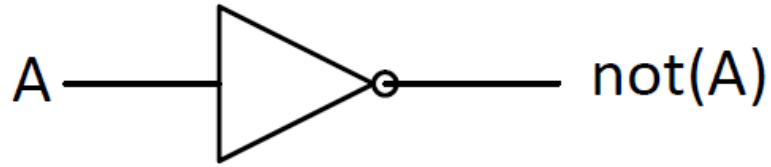
Con flujo de control => Input == 1 → Flujo por salida 1 => Output 1

Transistor

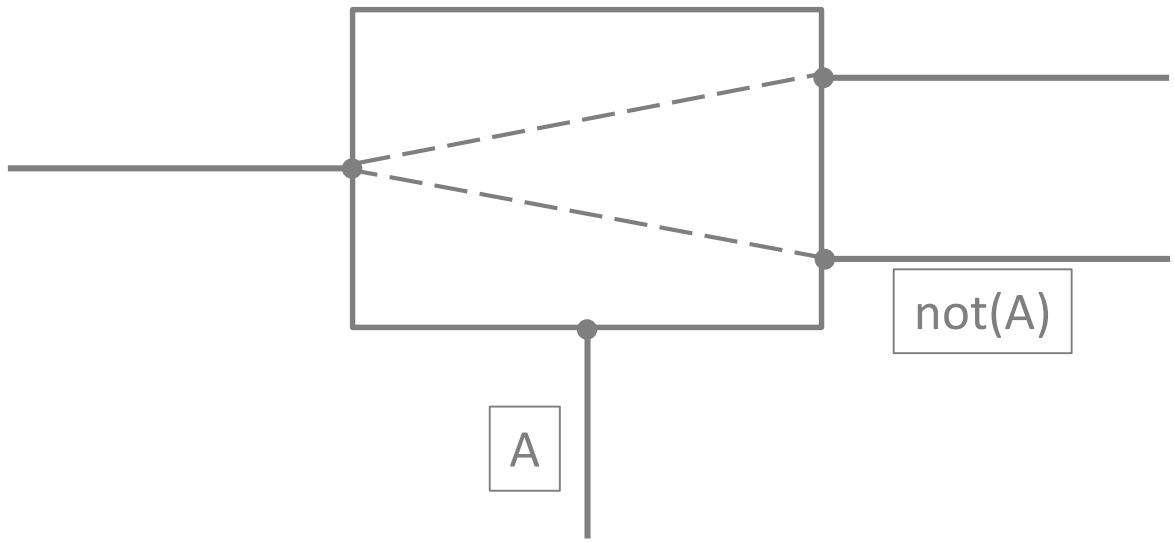


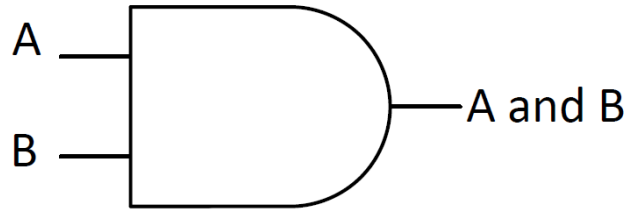
B == 0 → Sin flujo por E

B == 1 → Flujo por E

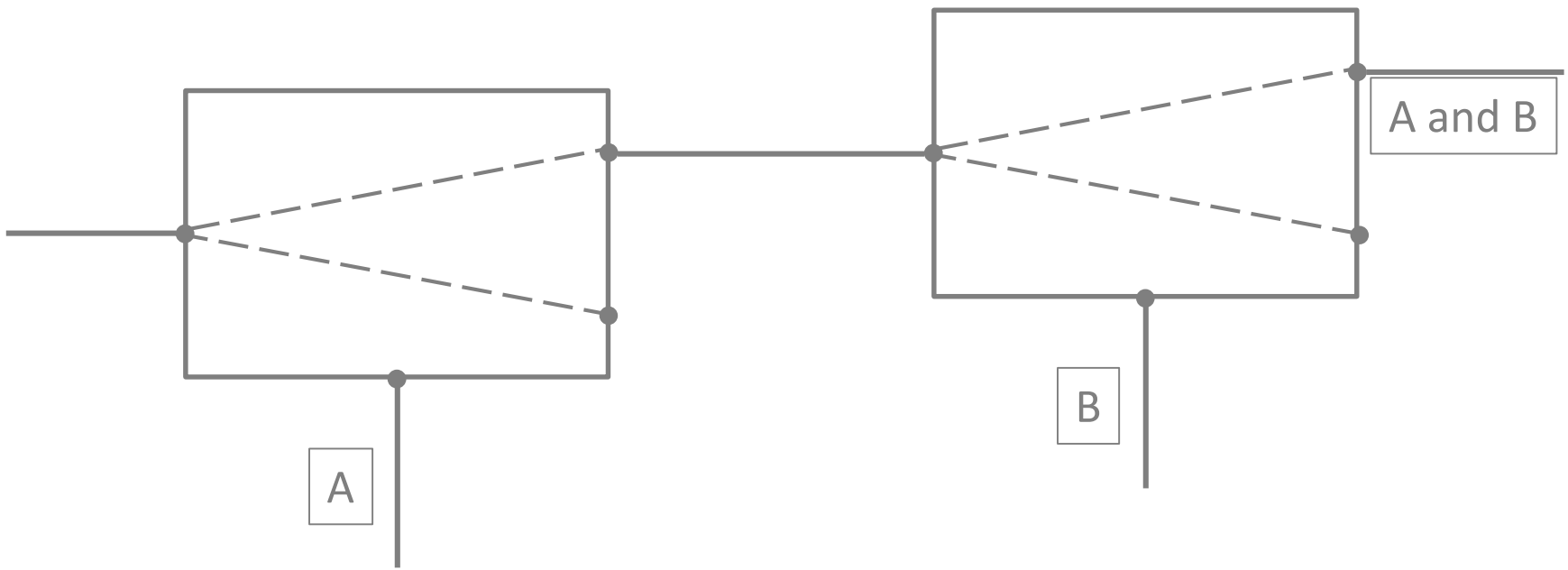


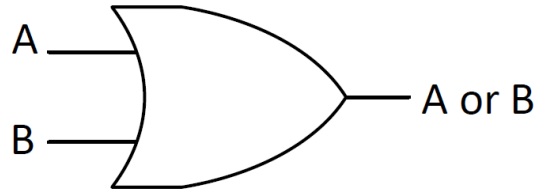
A	not(A)
0	1
1	0



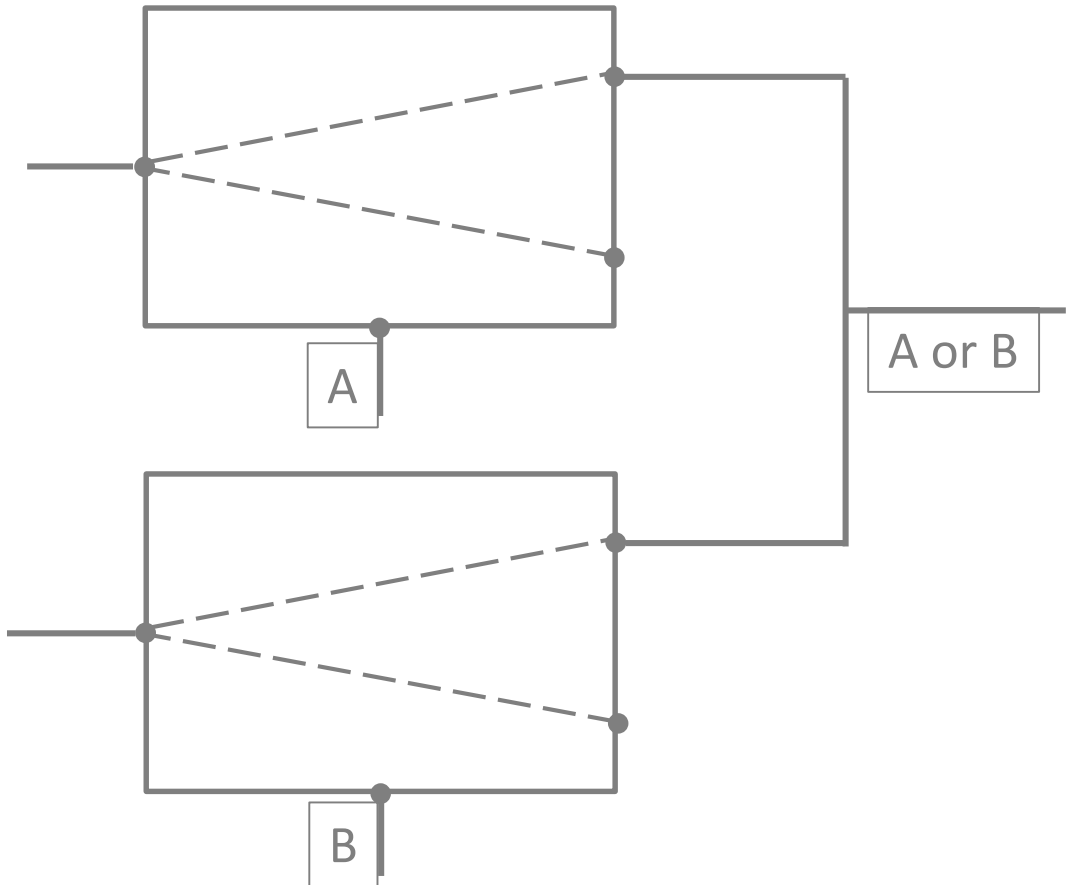


A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1



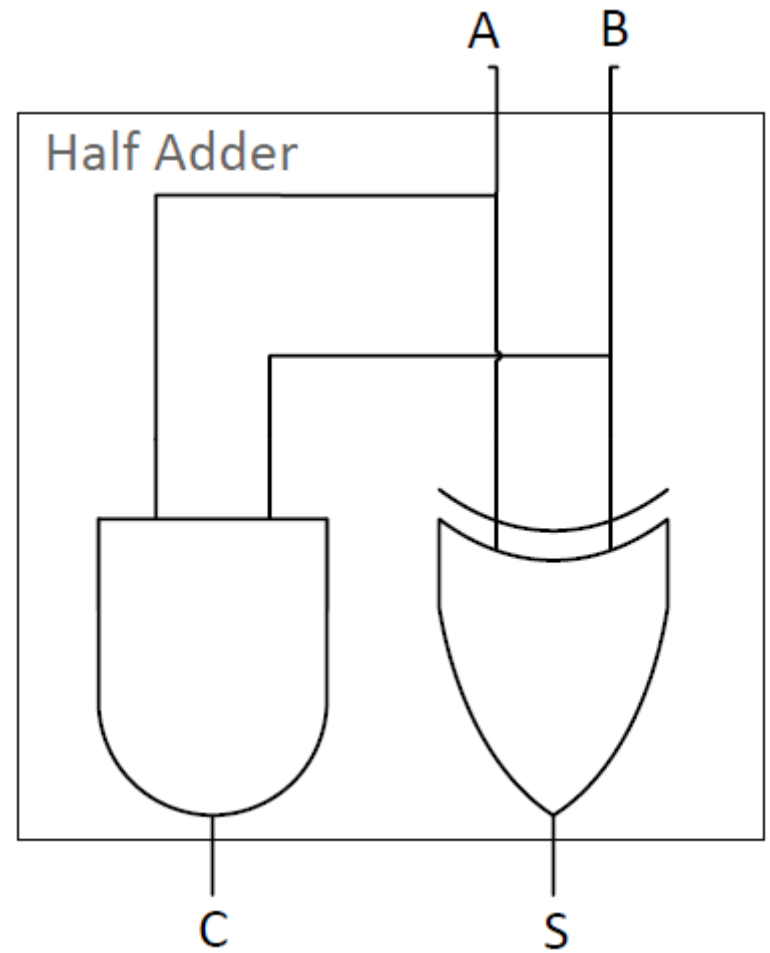


A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1



A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- $S = A \oplus B$
- $C = A \wedge B$

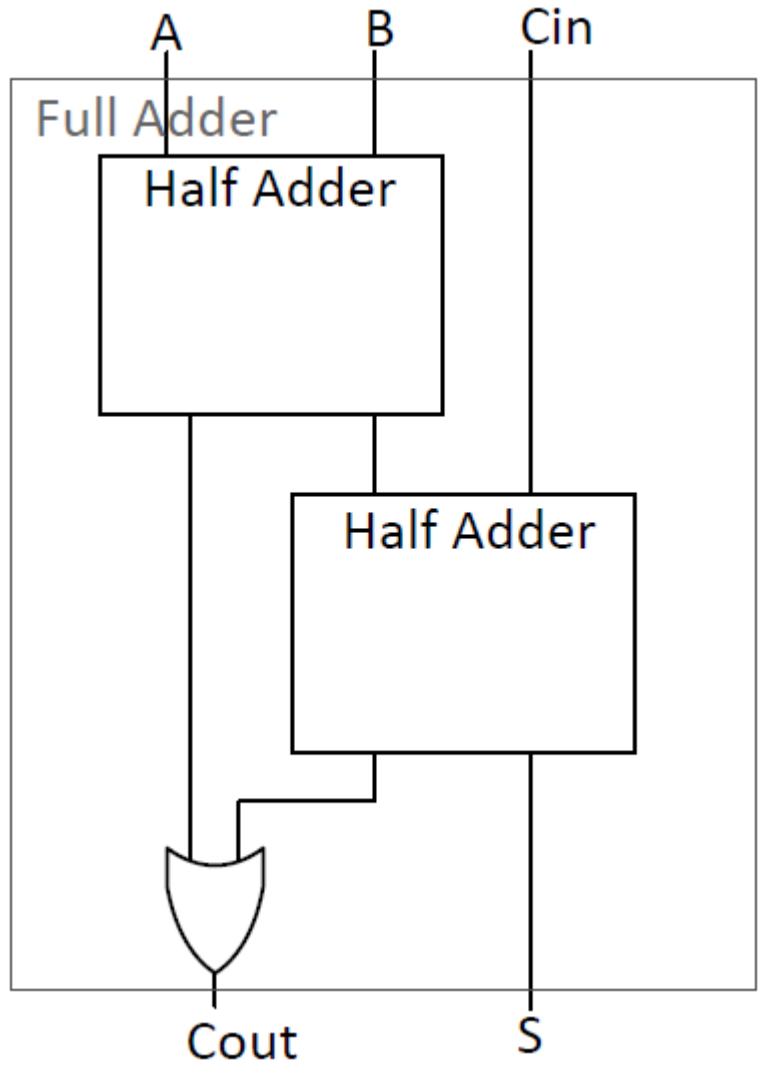


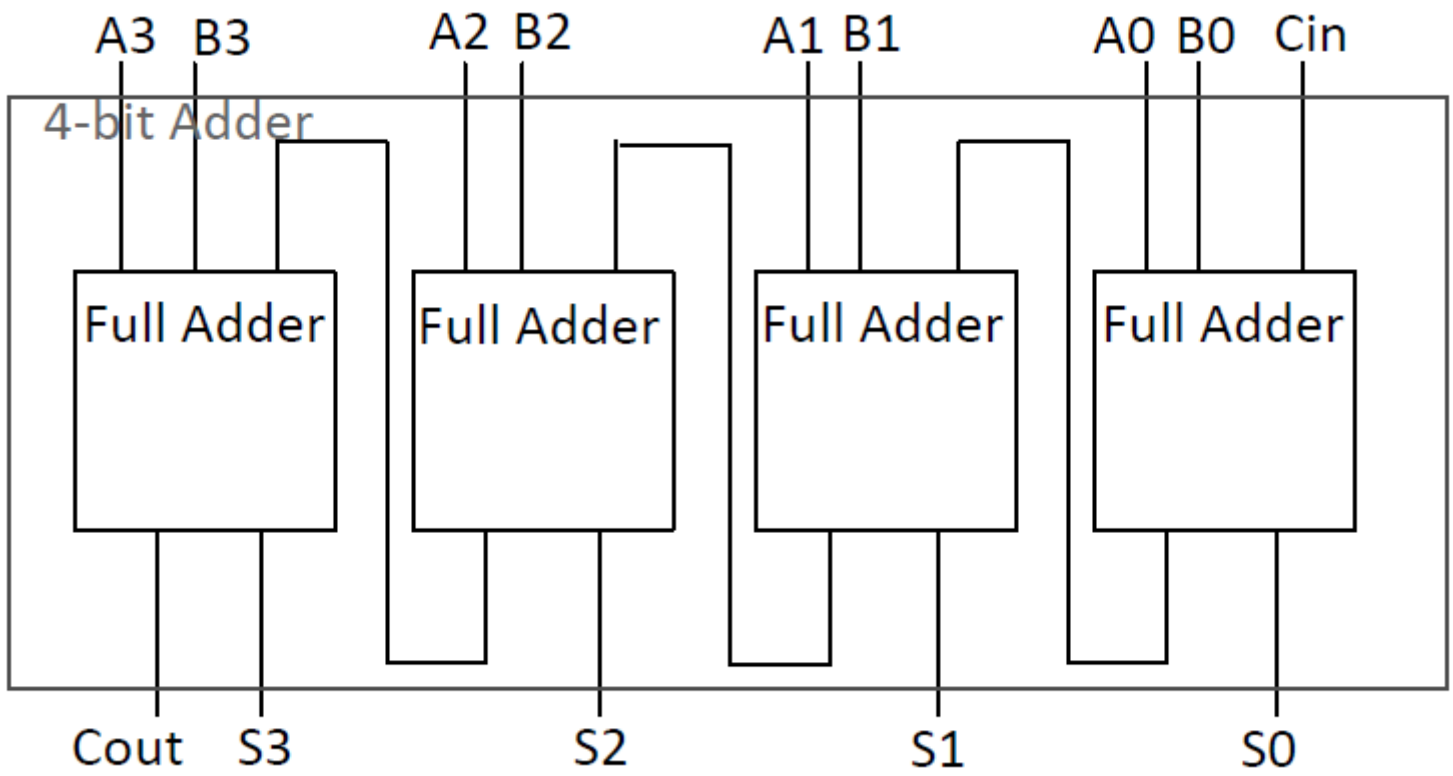
La clave consiste en tener módulos combinables y expandibles

- Aprovechemos la versatilidad de la lógica booleana y ampliamos el sumador a 4 bits.

- ¿Sirve poner 4 half-adders en paralelo?

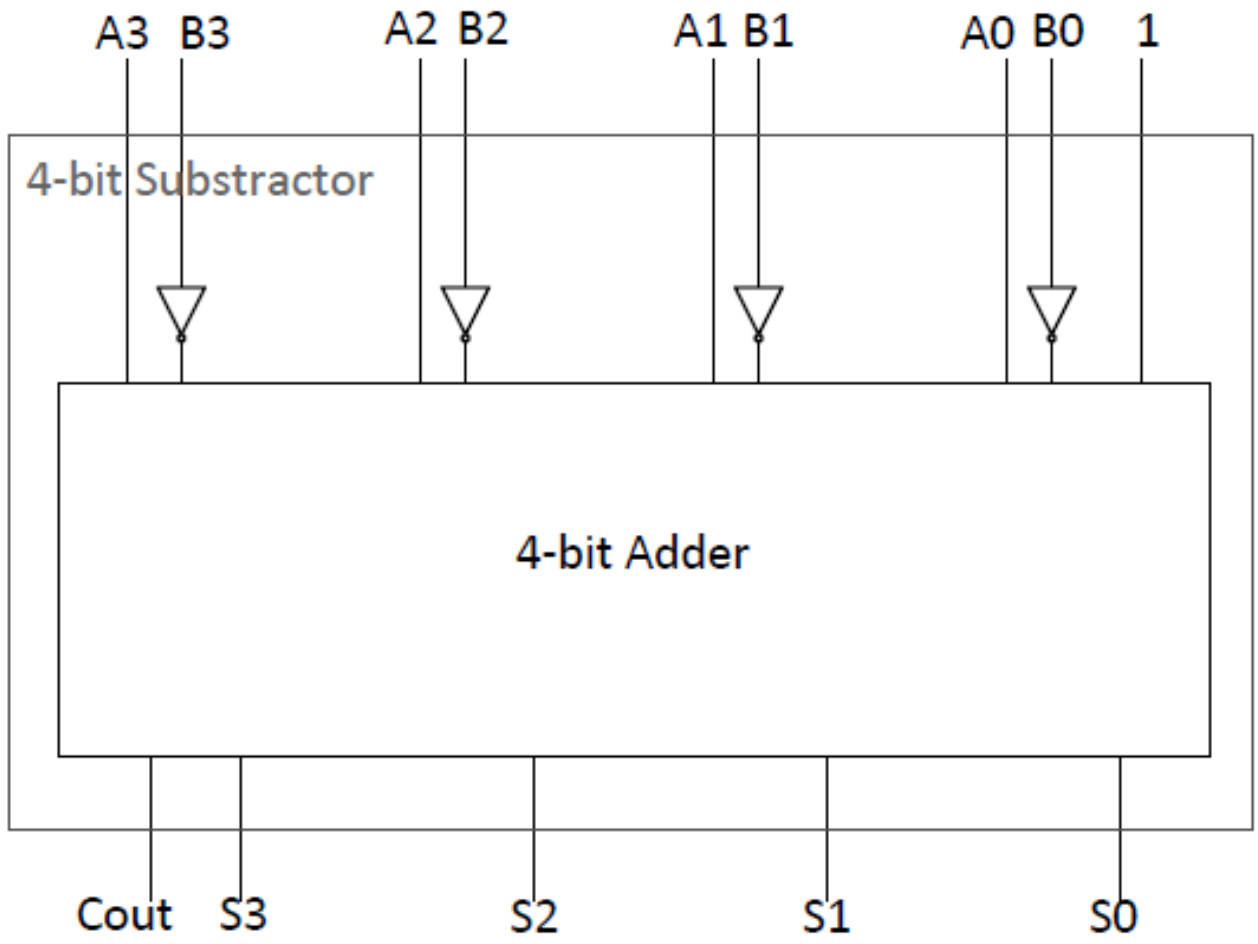
- Primero debemos construir un full-adder, que considere el carry de una operación anterior.


A
B
Cin
Full Adder
Half Adder
Half Adder
Cout
S


A3
B3
A2
B2
A1
B1
A0
B0
Cin
4-bit Adder
Full Adder
Full Adder
Full Adder
Full Adder
Cout
S3
S2
S1
S0

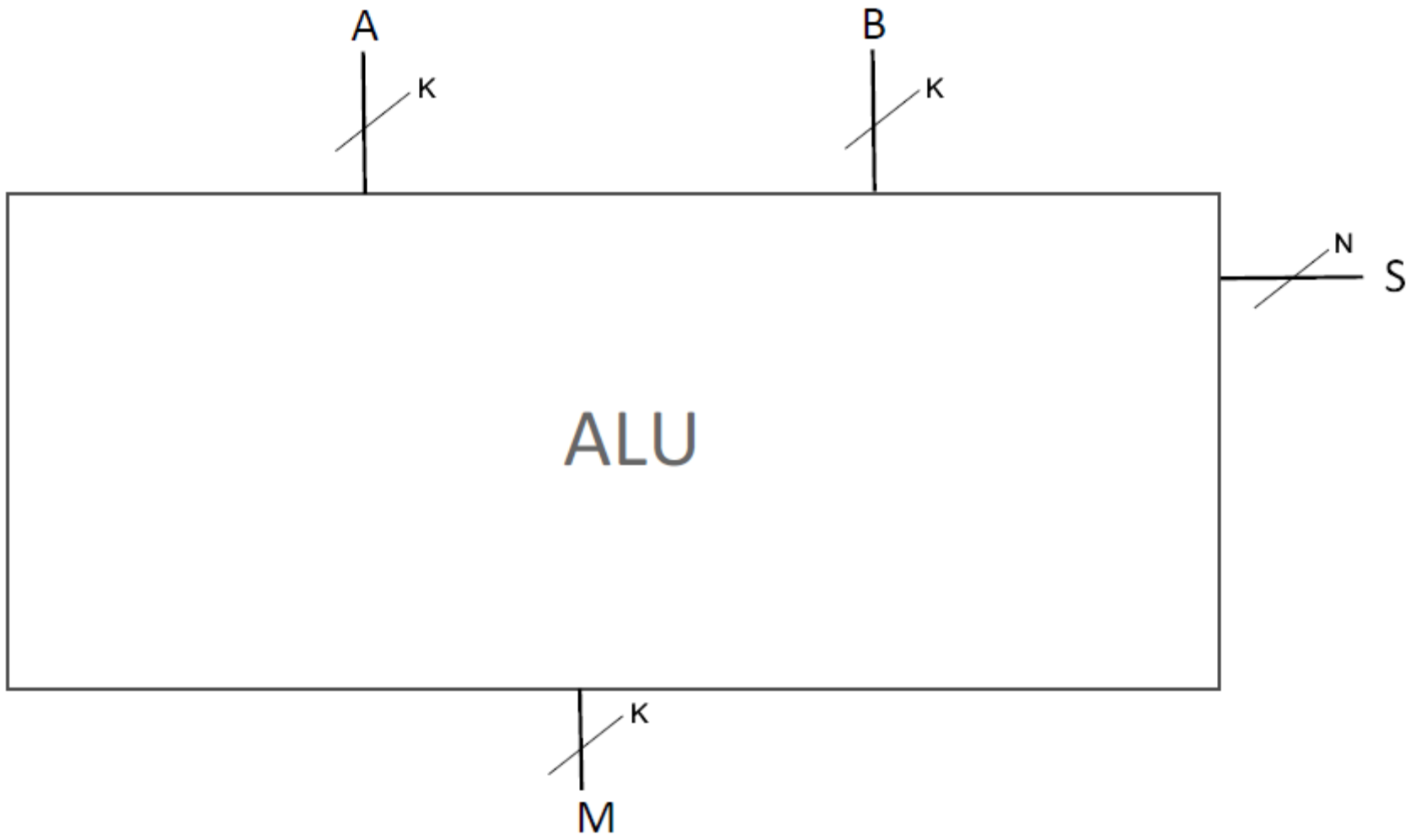
La clave consiste en tener módulos combinables y expandibles

- ¿Qué debemos hacer para diseñar un restador?
- Afortunadamente, no necesitamos mucho más.
- Aprovechando el complemento a 2, sumamos al minuendo el inverso aditivo del sustraendo y además ponemos en 1 el carry-in.


A3
B3
A2
B2
A1
B1
A0
B0
1
4-bit Substractor
4-bit Adder
Cout
S3
S2
S1
S0

El centro de operaciones de un computador

- El objetivo de esta unidad es acercarse lo más posible a la noción de un computador, desde el punto de vista de las operaciones.
- Un computador realiza principalmente cálculos, por lo que es fundamental construir una unidad eficiente para esto.
- El centro de operaciones de un computador es la unidad aritmética lógica (ALU), por la cual pasan todos los datos.


A
K
B
K
N
S
ALU
K
M

Enfrentemos primero las operaciones aritméticas

- Nuestro primer paso será un sumador/restador de 4 bits

- ¿Podemos construir esto usando sólo los elementos vistos hasta ahora en clases?

Elementos de control son necesarios

- Para diseñar un el sumador/restador aún nos faltan algunos elementos lógicos.

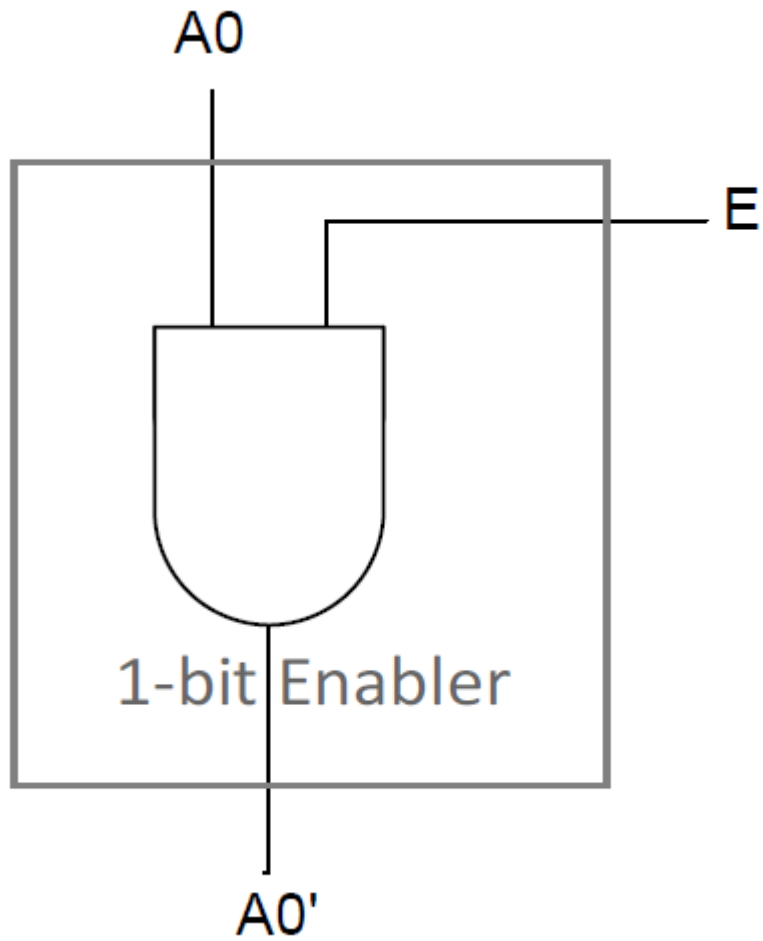
- Estos tienen relación con el control y la selección de las salidas y operaciones que se realizan.

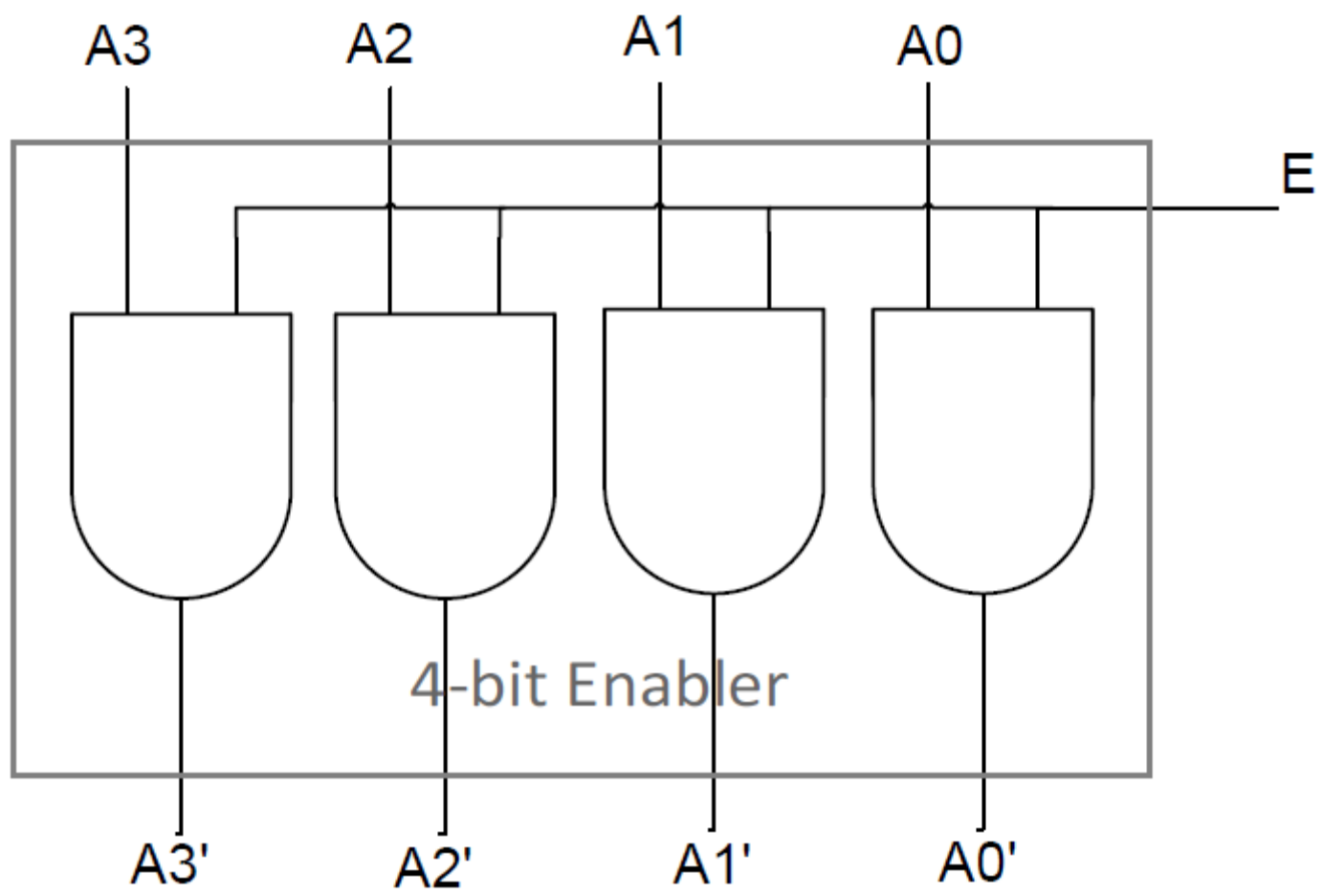
- Estos elementos se conocen como Enablers y Multiplexores o Mux.

Buses otorgan simplificación en el diseño de los componentes

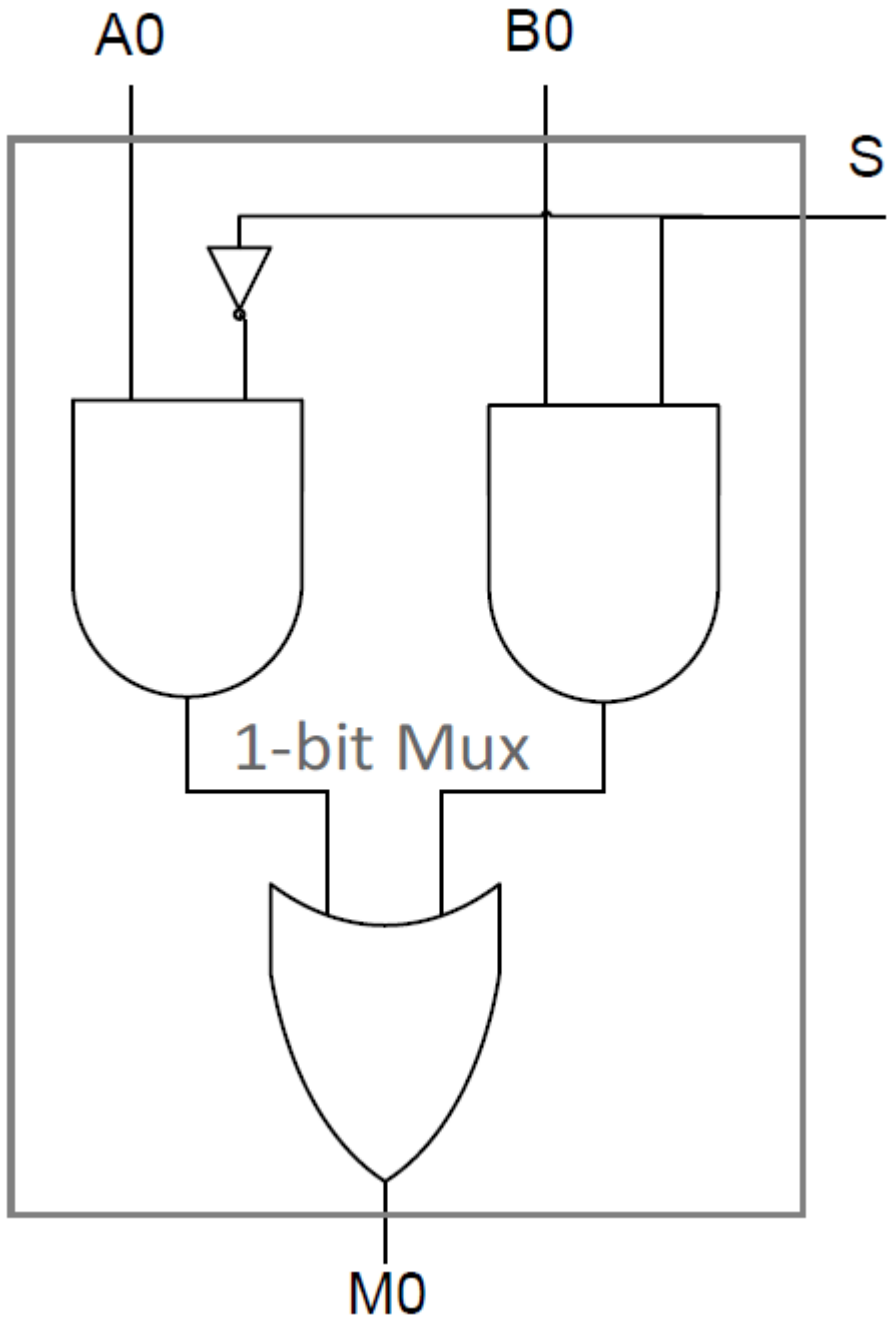
- Las conexiones usadas anteriormente sólo transmitían datos, ej. sumador.
- Ahora, además de eso, necesitamos transmitir órdenes de control.
- Así, se hace la distinción entre buses de datos y buses de control.

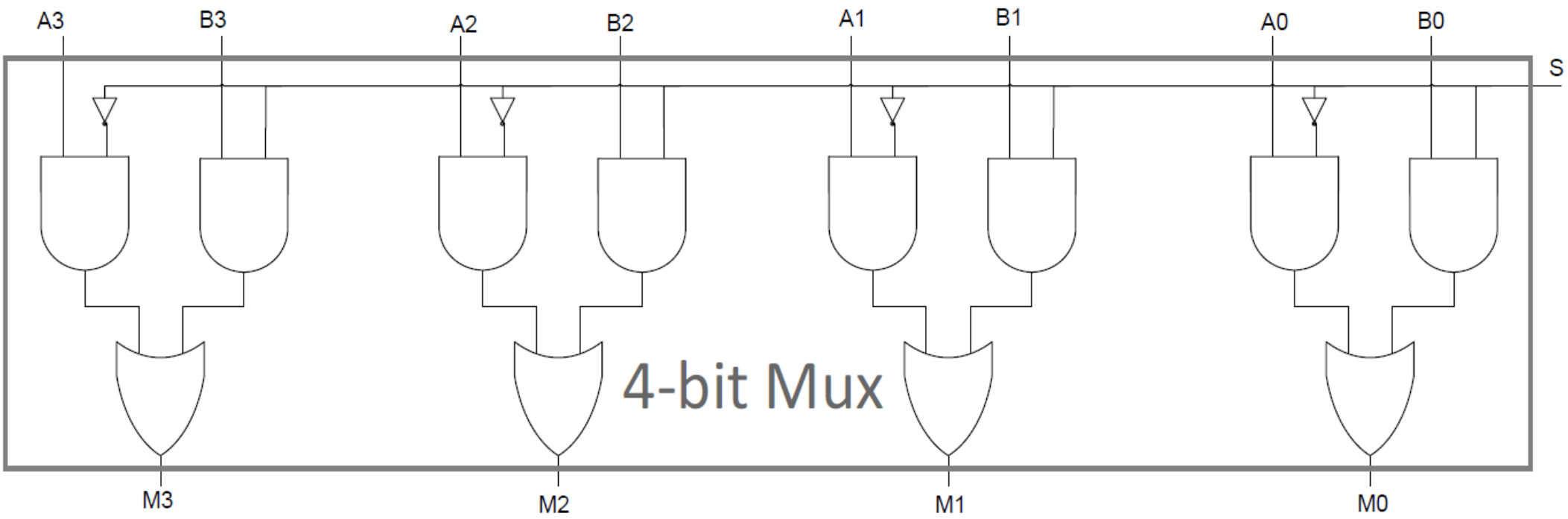
E	A0’
0	0
1	A0



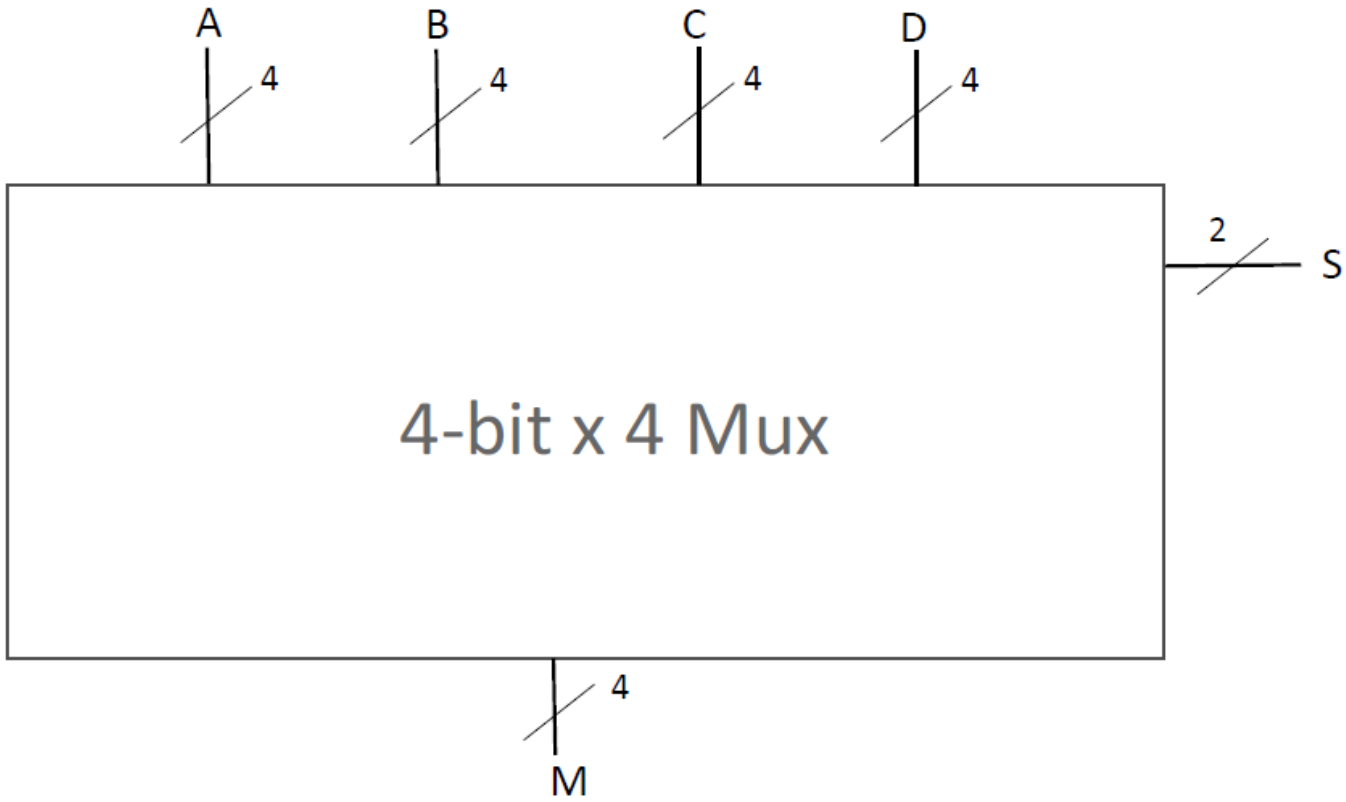

A3
A2
A1
A0
E
4-bit Enabler
A3'
A2'
A1'
A0'

S	M0
0	A0
1	B0

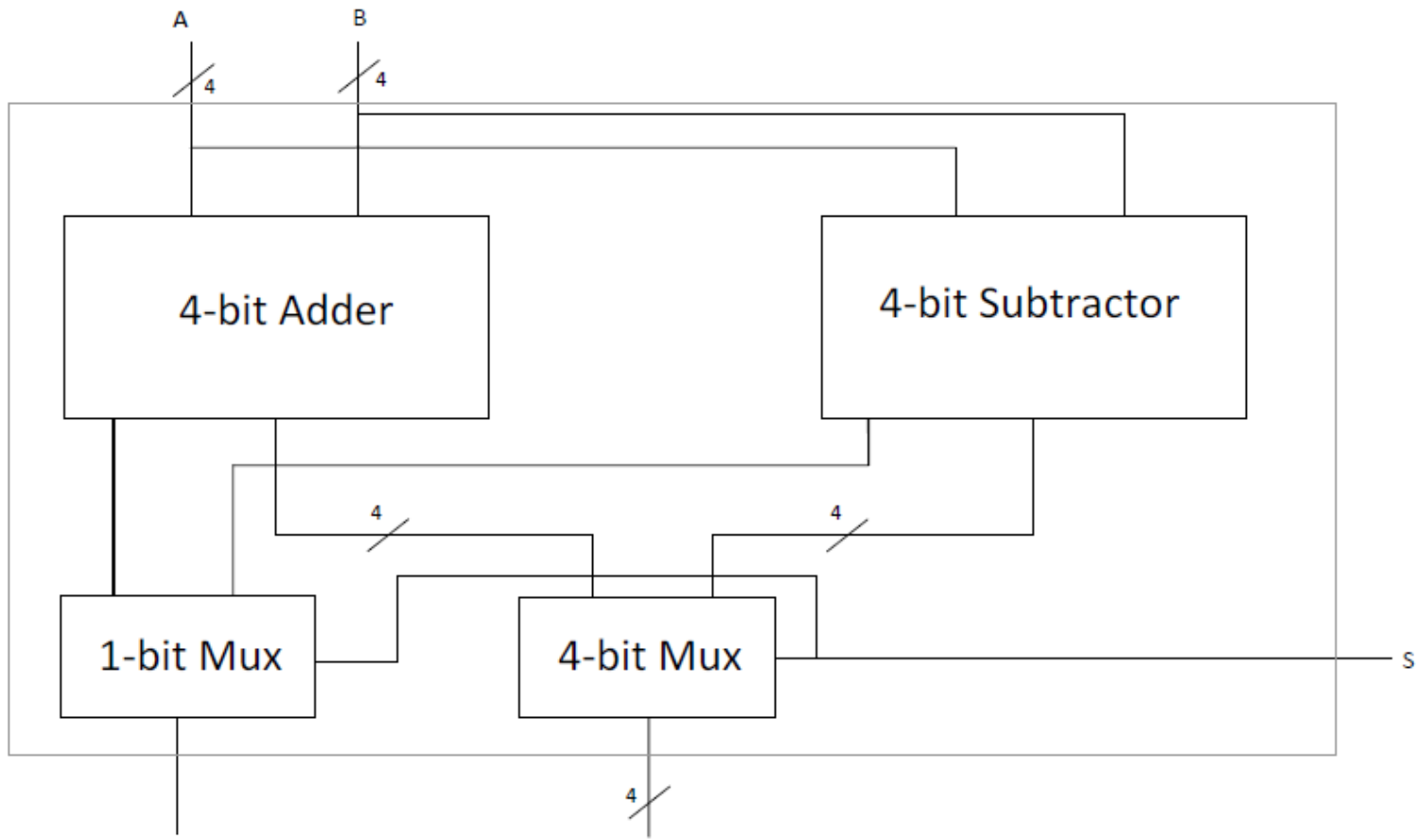


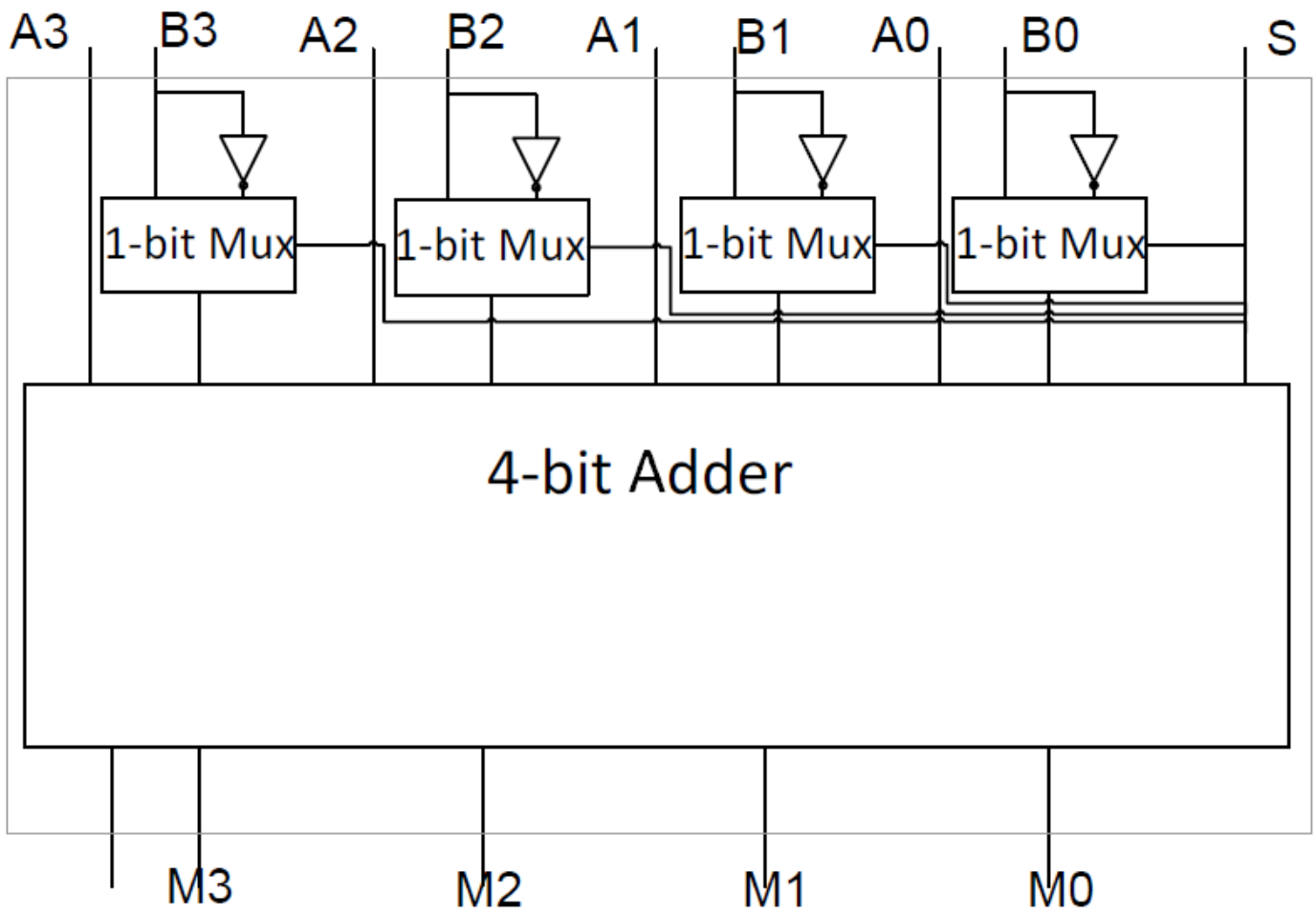

A3
B3
A2
B2
A1
B1
A0
B0
S
4-bit Mux
M3
M2
M1
M0

S1	S1	M
0	0	A
0	1	B
1	0	C
1	1	D

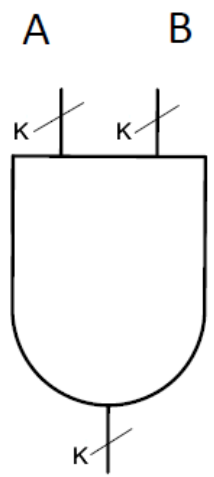


Cómo queda entonces el sumador/restador

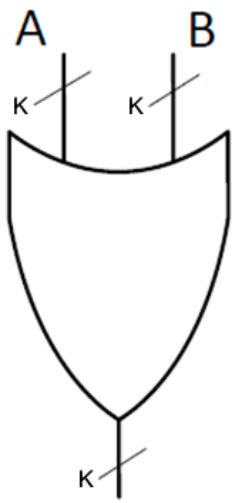



A3
B3
A2
B2
A1
B1
A0
B0
S
1-bit Mux
1-bit Mux
1-bit Mux
1-bit Mux
4-bit Adder
M3
M2
M1
M0

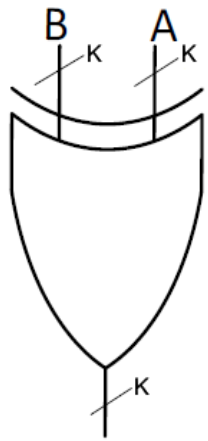
Operaciones lógicas *bitwise*



A and B



A or B



A xor B

not(A)

Sólo falta definir el ¿shifting?

- Una operación muy ocupada es el shifting o desplazamiento, que consiste en mover todas las cifras a la izquierda o la derecha.
- Ej: shift_left(0101) = 1010.
- ¿Por qué es esto importante?
- En binario, desplazar a la izquierda es análogo a multiplicar por 2, mientras que desplazar a la derecha es dividir por 2.
- ¿Qué pasa acá?: shift_left(1101) = ?
- Si no queremos perder cifras, utilizamos *rotate*.
- Ej: rotate_right(0101) = 1010.

Ya tenemos todos los elementos necesarios para construir la ALU

- Con todas las operaciones anteriores definidas, tenemos suficiente para completar una ALU de K bits con 8 operaciones.

S2	S1	S0	M
0	0	0	Suma
0	0	1	Resta
0	1	0	And
0	1	1	Or
1	0	0	Not
1	0	1	Xor
1	1	0	Shift left
1	1	1	Shift right

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores

Operaciones Aritméticas y Lógicas

Profesor: Hans Löbel